Univerza v Ljubljani
Fakulteta za računalništvo in informatiko

Matic Podpadec

Sistem za uravnavanje svetlobe v prostoru

diplomsko delo

visokošolski strokovni študijski program
prve stopnje
računalništvo in informatika

Mentor: doc. dr. Rok Rupnik

Ljubljana, 2017

Besedilo je oblikovano z urejevalnikom besedil LaTeX.

Fakulteta za računalništvo in informatiko izdaja naslednjo nalogo:

Tematika naloge:

Osvetljevanje prostorov je relativno kompleksno področje, saj lahko končni učinek kvalitete in atraktivnosti osvetlitve dosežemo na več načinov, a ob precej različnih stroških osvetlitve. Zasnujte sistem za upravljanje svetlobe v prostoru. Sistem naj vključuje tako senzorje za avtomatski nadzor kot tudi možnost ročnega nadzora preko aplikacije. Za strežnik uporabite Raspberry 3 (model B), za programski jezik uporabite Pyton, uporabite pa tudi ogrodja Django in Django REST.

Za pomoč pri diplomskem delu bi se zahavlil mentorju doc. dr. Roku Rupniku in družini za podporo.

Svoji družini.

Kazalo

Seznam uporabljenih kratic

kratica	angleško	slovensko
API	Application programming interface	Vmesnik za namensko programiranje
LED	Light Emiting Diode	Svetleča dioda
MVC	Model View Controller	Model pogled kontroler
PWM	Pulse-width modulation	Modulacija impulzne širine
IoT	Internet of Things	Internet stvari
IEA	International Energy Agency	Mednarodna organizacija za energijo
IDE	Integrated development environment	Integrirano razvojno okolje

Povzetek

Naslov: Sistem za uravnavanje svetlobe v prostoru

Avtor: Matic Podpadec

V diplomskem delu smo se lotili izdelave sistema za upravljanje svetlobe v prostoru. Namen sistema je nadzor virov svetlobe v prostoru s pomočjo senzorjev za avtomatski nadzor in aplikacije za ročni nadzor. Za reševanje problem smo uporabili Raspberry 3 model B, ki nam služi kot strežnik za našo spletno aplikacijo in API, preko katerega poteka komunikacija z ostalimi moduli. Aplikacija in API sta napisani v programskem jeziku Python 3 s pomočjo ogrodij Django in Django REST. Podatki so shranjeni v podatkovni bazi SQLite3, nadzor nad strojno opremo izvajamo preko mikrokontrolerja Arduino UNO.

Ključne besede: računalnik, IoT, senzorji, Arduino, Raspberry Pi.

Abstract

Title: The System for the Management of Indoor Light

Author: Matic Podpadec

In this thesis we have created a system for indoor light management. The main purpose of this system is to control the light inside our home, office or any other indoor space. We've achieved this with the help of automatic light sensor as well as manual control of the application. The entire system is controlled by Raspberry Pi 3 model B, which functions both as a server for the web based application, as well as the API for communication with other modules. Both API and the application are written in Python 3 working with frameworks Django and Django REST. For data storage we have used SQLite3 and the main control over hardware is done by Arduino UNO.

Keywords: computer, IoT, sensors, Arduino, Raspberry Pi.

Poglavje 1

Uvod

1.1 Motivacija

Živimo v času, ko si težko predstavljamo življenje brez visoke tehnologije. To se vidi predvsem v hitrem razvoju tehnologij interneta stvari (IoT). Področje se ukvarja s stvarmi, ki jih preko senzorjev, aktuatorjev, programske opreme in podobnih tehnologij povežemo v omrežje. V tem jih lahko nato nadziramo s poljubne lokacije preko mobilnih naprav, ali kakšnih namenskih sistemov v primerih večjih IoT projektov, kot je nadzorovanje semaforjev mesta. Strokovnjaki predvidevajo, da bo do leta 2020 v omrežje povezano okoli 30 milijard tovrstnih naprav [22].

Za hiter razvoj področja interneta stvari sta glavna dva razloga. Prvi med njima je poenostavitev nadzora kompleksnih sistemov. Te sisteme je lažje tudi posodobiti, zamenjati ali odstraniti. Drugi je človeška želja po udobnejšem življenju. Ker se tehnologija hitro razvija so razni senzorji, potrebni za tovrstne sisteme, cenovno vedno bolj dostopni. Enako velja tudi za programsko opremo. Velika pobudnika za večje zanimanje o področju IoT sta tudi podjetje Arduino LLC in fundacija Raspberry Pi, ki sta s svojimi izdelki omogočila poceni izdelovanje prototipov.

Celotno področje interneta stvari je zelo široko, zato se bomo v diplomskem delu lotili samo področja pametne razsvetljave v prostoru. Razsvetljava

je namreč glede na poročilo IEA, med glavnimi porabniki električne energije s kar 19% skupne porabe elektrike in je krivec za 6% toplogrednih plinov [16].

1.2 Cilj

V tej diplomski nalogi se bomo lotili implementacije sistema za upravljanje svetlobe v prostoru, ki je v širšem pogledu del pametne razsvetljave. Poskusili bomo implementirati sistem, ki bo s pomočjo svetlobnih senzorjev in človeškega vnosa preko računalniškega sistema lahko nadziral razsvetljavo prostora. Poleg varčevalnih ukrepov bo imel sistem vgrajeno tudi funkcijo, ki nam bo priporočila osvetljenost, primerno za boljše počutje in zdravje.

1.3 Metodologije

Sistem bomo implementirali v več delih. Za branje podatkov senzorjev in podajanje naših ukazov svetilom bomo uporabili mikrokontroler Arduino UNO, ki ga bomo sprogramirali v programskem jeziku C++. Ukaze in rezultate bo mikrokontroler po serijski komunikaciji sprejel, oziroma posredoval majhnemu računalniku Raspberry Pi 3 model B. Na tem računalniku bomo poganjali strežnik, ki ga bomo napisali v programskem jeziku Python. Ta strežnik bo deloval kot API, s pomočjo katerega bomo lahko nadzorovali naše naprave. Komunikacija med APIjem in končnim uporabnikom poteka s pomočjo namenske aplikacije.

1.4 Zgradba diplomske naloge

Diplomska naloga je sestavljena iz 6 poglavij

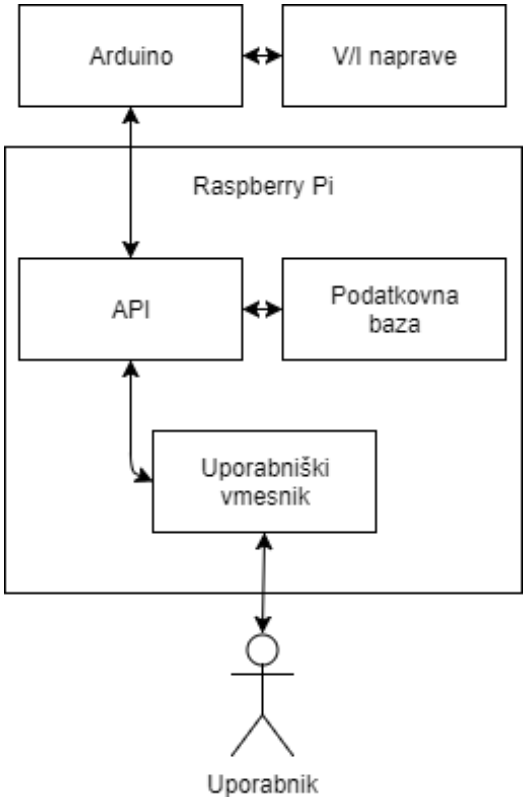
- Prvo poglavje je uvodno in predstavi namen diplomskega dela
- Drugo poglavje opisuje strojno in programsko tehnologijo uporabljeno tekom diplomskega dela

- Tretje govori o vplivu svetlobe na zdravje in počutje
- Četrto poglavje opiše implementacijo našega sistema
- Peto poglavje je namenjeno možnim nadgradnjam sistema
- Šesto poglavje je sklep

Poglavje 2

Analiza zahtev

Pred začetkom dela smo morali najprej analizirati zahteve našega sistema. V ta namen smo naredili grobo skico modulov, ki jih bomo potrebovali. Kot je razvidno na sliki 2.1 potrebujemo skupaj štiri ločene module, na katere so priklopljene še vhodne in izhodne naprave. Prvi modul predstavlja programsko in strojno opremo na najnižjem nivoju. V njem bodo vključeni senzorji, svetila, mikrokontroler in programska koda, ki ta mikrokontroler nadzira. Naslednji del, ki ga moramo zasnovati je podatkovna baza. Znotraj te se bodo nahajali vsi pomembni podatki potrebni za delovanje. Za komunikacijo s končnim uporabnikom potrebujemo tudi uporabniški vmesnik. Ker nočemo, da bi bili z uporabniškim vmesnikom omejeni na samo določene platforme, smo se odločili sprogramirati še API, ki bo med seboj povezoval prej omenjene module. Tako bomo dosegli največjo neodvisnost posameznih modulov, kar nam bo omogočilo preprosto nadgrajevanje ali posodabljanje našega dela.

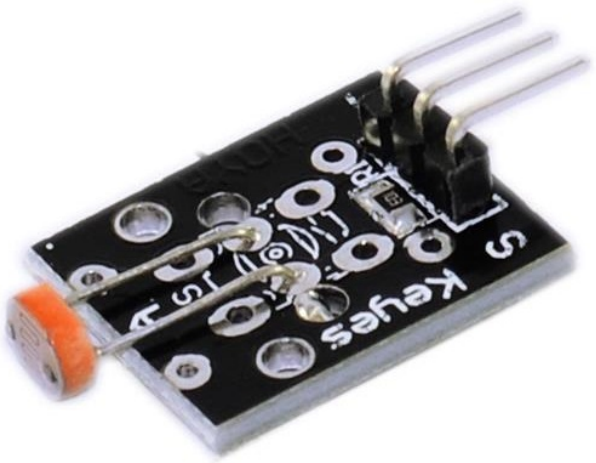


Slika 2.1: Gradniki našega sistema.

2.1 Strojna oprema

Prvi modul, ki smo ga definirali, je strojna oprema. Glavni del strojnega dela te naloge je mikrokrmilnik. Na trgu je že precej izbire, za prototipiranje se nam je zdel najbolj primeren Arduino UNO. Mikrokontroler ima nizko ceno in vsebuje več analognih in digitalnih vhodno-izhodnih točk. Sledile so vhodne in izhodne naprave. Ker je cilj diplomskega dela le prototip sistema, smo se odločili, da je za simulacijo svetilk dovolj navadno svetilo LED. Za senzorsko enoto smo se odločili uporabiti samo fotocelice Arduino KY-018 (slika 2.2).

Celica pride z vgrajenim dodatnim uporom, preko katerega beremo trenutno izmerjeno vrednost. Zadnji del strojne opreme, ki jo potrebujemo,



Slika 2.2: Fotocelica Arduino KY-018

je računalnik, preko katerega bo potekala komunikacija s končnim uporabnikom. Zaradi želje po nizkocenovnem in varčnem računalniku smo izbrali Raspberry Pi računalnik.

2.1.1 Arduino UNO

Arduino UNO je mikrokontroler narejen na osnovi ATmega328P [3]. Sestavlja ga 14 digitalnih vhodno-izhodnih točk, 6 od teh je primernih za uporabo PWM, 6 analognih vhodno-izhodnih točk, 16MHz kremenčev kristal, USB povezavo in tipko za ponovni zagon [3]. Ploščica je izšla skupaj s programsko opremo Arduino IDE. Trenutno je najbolj podprt, dokumentiran in najbolj robusten Arduinov izdelek. V primeru da med izdelavo prototipa poškodujemo ploščico je potrebno zamenjati samo čip, ki je zelo poceni.

Lastnosti Arduino UNO [3]:

- Mikrokontroler ATmega328P
- Delovna napetost 5 V
- Napajanje (priporočeno) 7 V do 12 V

Slika 2.3: Arduino UNO.

- Napajanje (omejitve) 6V do 20 V
- 14 digitalnih V/I nožic
- šest PWM digitalnih V/I nožic
- šest analognih vhodnih nožic
- DC tok an V/I nožico 20 mA
- DC tok za 3.3 V nožico 50 mA
- Spomin flash 32 KB (ATmega328P)
- SRAM 2 KB (ATmega328P)
- EEPROM 1 KB (ATmega328P)
- Hitrost ure 16 MHz

2.1.2 Raspberry Pi 3 model B

Raspberry Pi je mikroračunalnik, izdelan v Združenem kraljestvu za potrebe poučevanja računalništva [18, 24, 17]. Odločili smo se se za uporabo Raspberry Pi 3 Model B (slika 2.4), ki ga bomo uporabili kot strežnik našega sistema, saj je ta močnejši od predhodnikov, vendar vseeno ostaja cenovno dostopen.

Slika 2.4: Raspberry Pi 3 Model B.

Lastnosti RPi 3 Model B [12]:

- Quad Core 1.2 GHz Broadcom BCM2837 64bit CPE
- 1 GB RAM
- BCM43438 brezžični LAN in Bluetooth Low Energy (BLE) na plošči
- 40-nožični razširjen GPIO
- 4 USB 2 vhoda

- 4 Polni stereo izhod in kompozitni video priključek
- HDMI priključek
- Vhod za kamero CSI, namenjen povezavi z Raspberry Pi kameri
- Vhod za zaslon DSI, namenjen Raspberry Pi zaslonu na dotik
- Vhod za kartico Micro SD za OS ter shranjevanje podatkov

2.2 Programska oprema

Po dokončni odločitvi glede izbire strojne opreme, smo se lotili analize programskega dela. Na izbranem strežniku bo več modulov programske opreme. Najprej bo potrebno napisati API namenjen komunikaciji med našo aplikacijo, podatkovno bazo in mikrokontrolerjem. API mora biti dovolj dovršen, da sam obdela večino podatkov in mikrokontrolerju pošilja le minimalno število ukazov. Drugi del je aplikacija, ki je namenjena komunikaciji med končnim uporabnikom in API modulom. Za ta dva modula smo se odločili uporabiti programski jezik Python in okvir Django ter Django REST. Odločili smo se, da bomo za začetek spletne aplikacije uporabili tudi tehnologije HTML5, CSS, Bootstrap in Javascript.

2.2.1 Arduino IDE

Arduino IDE je „cross-platform“ aplikacija napisana v programskem jeziku Java, ki je zasnovana na podlagi aplikacij Processing in Wiring [1]. Vsebuje vse osnovne funkcije urejevalnikov besedil kot sta kopiraj in prilepi ter funkcije pomembne za lažje programiranje, kot so poudarek sintakse, ujemanje oklepajev in samodejno zamikanje kode. Poleg funkcij urejevalnika besedil ima na voljo tudi nekaj razvijalskih orodij. Od teh je za nas glaven „Serial Monitor“ s katerim smo testirali naše funcije, preden smo nadaljevali z razvojem sistema.

2.2.2 PyCharm

PyCharm je IDE podjetja JetBrains. Namenjen je predvsem programiranju v programskem jeziku Python, poleg tega pa vsebuje podporo za mnoge spletne tehnologije, med njimi tudi za ogrodje Django, ki ga bomo uporabljali. IDE je zelo napreden in nam omogoča izredno dober pregled nad celotnim projektom. Med programiranjem si lahko pomagamo s pametnim urejevalnikom kode, pametno navigacijo po kodi in izjemno inteligentnim samodejnim dokončevanjem sintakse. Prav tako je sposoben naprednega preoblikovanja kode [11].

2.3 Uporabljene tehnologije

2.3.1 Programski jezik C/C++

Za programiranje mikrokrmilnika smo se odločili uporabiti programski jezik C++. Sicer bi nam najbolj precizno nadziranje sistema omogočil mikro C, vendar ta žal ni primeren za prototipiranje. C++ je med najbolj priljubljenimi splošno namenskimi programskimi jeziki. Prevede se v strojni jezik, kar pomeni da se izvaja hitreje kot interpretirani jeziki kot sta Python in Java. Zaradi zahteve po bolj podrobnem programiranju, je hkrati tudi bolj varčen pri porabi elektrike. Glavni razlog, da smo se odločili za njegovo uporabo je Arduino IDE, ki je uradno razvijalno okolje za Arduino mikrokrmilnike. Razlika med splošno kodo C++ in kodo Arduino sketch je ta, da slednja zahteva dve glavni funkciji „setup“, za začetno postavitev in „loop“, ki deluje kot „main“, le da se izvaja v nedogled [2].

2.3.2 Python

Za razvoj glavnega dela naše aplikacije, to je API, uporabimo programski jezik Python. Jezik je večnamenski z visoko podporo za veliko različnih področij. Narejen je tako, da je v prednosti vedno berljivost kode, kar je pri večjih projektih zelo pomembno. Prav tako nam razne knjižnice, ki jih

skupnost ponuja, omogočajo preprosto implementacijo raznoraznih funkcij, s katerimi postane naša koda preglednejša. Tekom projekta smo se odločili za uporabo ogrodja Django in Django REST.

2.3.3 Ogrodje Django

Django je odprtokodno ogrodje za programski jezik Python. Njegov glavni namen je olajšanje pisanja spletnih aplikacij. Poleg mnogih funkcionalnosti je izjemno stabilen, poskrbi tudi za varnost naših aplikacij. Django deluje na treh nivojih imenovanih model, pogled (ang. view) in predloga (ang. template). Vsak od njih ima v ogrodju svojo datoteko z vsebino. Ta arhitektura je sorodna bolj znani arhitekturi MVC. Modeli so namenjeni za delo s podatkovno bazo. Znotraj „models.py“ definiramo vse naše tabele, njihove atribute in omejitve v obliki razreda z metodami za obdelavo teh podatkov. Ogrodje nato z ukazi „makemigrations“ in „migrate“ ustvari podatkovno bazo glede na razrede v podatkovni bazi, ki si jo izberemo. „Views.py“ aplikaciji pove katero stran in s kakšnimi podatki mora prikazati glede na zahtevo uporabnika. Preden lahko kličemo funcijo, ki je del pogleda, jo moramo še definirati v „urls.py“. Zadnji del arhitekture so predloge. Te so v bistvu datoteke HTML, ki se napolnejo s podatki dobljenimi v „views.py“ preko modelov.

2.3.4 Ogrodje Django REST

REST je način komunikacije med napravami v omrežju. Spletne storitve, ki podpirajo REST omogočajo dostop in manipulacijo besedilne predstavitve spletnih virov z uporabo poenotenih in v naprej določenih brezstanjskih operacij [20]. Besedilna predstavitev uporabljena v tem diplomskem delu je formata JSON, sicer pa bi lahko uporabili tudi formate kot sta XML ali celoten HTML. Ker želimo za naš sistem napisati tudi API, smo se odločili uporabiti tudi ogrodje Django REST. Ta nam omogoča preprosto implementacijo spletnega vmesnika za posredovanje podatkov na različna okolja preko poljubnega formata. Kot smo že omenili, bo to v tem diplomskem delu

JSON.

Uporabnosti REST ogrodja:

- poenostavljena izdelava API,
- avtentikacija OAuth1a in OAuth2,
- serializacija podatkov.

2.3.5 JSON: JavaScript Object Notation

JSON je format za shranjevanje in izmenjavo podatkov v navadnem besedilu [8]. Kadar govorimo o pošiljanju podatkov med strežnikom in brskalnikom, oziroma drugimi končnimi napravami preko spleta, je najbolj preprost in učinkovit način čistopis, za kar je JSON zelo primeren format. Višji jeziki prav tako v večini podpirajo „serializatorje“, ki lahko format JSON pretvorijo v obliko, primerno za obdelavo v aplikaciji.

2.3.6 HTML, CSS, Bootstrap

Spletna aplikacija mora v današnjem svetu zgledati tudi privlačno, saj je sicer nihče ne želi uporabiti. Ogrodje Django uporablja poglede, ki bodo v osnovi napisani s standardnim označevalnim jezikom HTML [25]. Med značkami HTML pa se bodo znašle tudi značke ogrodja Django. Za lepšo obliko bomo poskrbeli z datotekami CSS [6] in ogrodjem Bootstrap [5].

Poglavje 3

Vpliv svetlobe na zdravje in počutje

Znanost že dolgo raziskuje tudi področje vpliva svetlobe na okolje in ljudi. Skozi leta raziskav so ugotovili, da različna svetloba skozi dan na ljudi vpliva na zelo raznolike načine. Lotili so se tako raziskav glede fizičnega, kot tudi psihičnega zdravja. Ker se naše diplomsko tiče samo vidne svetlobe, bomo opisali nekaj škodljivih, oziroma pozitivnih učinkov le-te svetlobe.

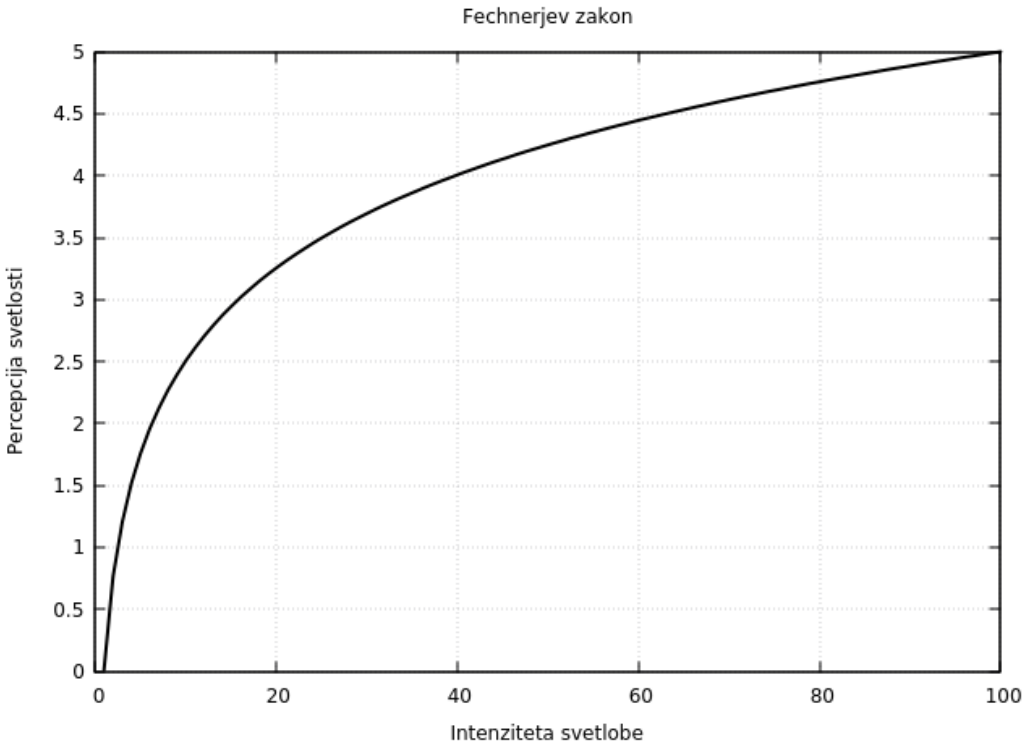
Naše telo se slabo odziva predvsem na umetno nočno svetlobo. Že več raziskav je povezalo višjo prisotnost raka pri ljudeh, ki delajo v nočni izmeni. Poleg raka so nočno svetlobo povezali tudi z drugimi nočnimi boleznimi, kot sta na primer prekomerna teža in diabetes [21]. Presvetla svetloba ponoči prav tako bolj uničuje naš vid, enako velja za dolgotrajno uporabo prenizke svetlobe. Iz teh ugotovitev lahko predpostavimo, da na naše zdravje ne vpliva samo moč, temveč tudi tip svetlobe. V članku „Blue Light has a dark side“ lahko preberemo, da je pomemben dejavnik modra svetloba [4]. Ta je danes zelo pogosto uporabljena, saj skupaj z zeleno in rdečo v elementih LED sestavlja belo svetlobo. Čeprav modra svetloba ljudem zelo škoduje v nočnem času, nam prav tako pomaga čez dan. Modra svetloba je danes uporabljena v večini moderne tehnologije. Znanstveniki Univerze v Torontu priporočajo, da v nočnem času pri uporabi modernih naprav, ki svetijo tudi

z modro svetlobo, uporabljamo posebna očala [4]. Ta naj filtrirajo svetlobo tako, da modra svetloba na nas ne vpliva več. Če se že ne moremo izogniti nočni razsvetljavi je priporočljiva zatemnjena rdeča svetloba.

Tako kot fizično, se s svetlobo spreminja tudi naše psihično stanje. Z raziskavami so dokazali, da nočna svetloba negativno vpliva na človeško počutje. 20% ljudi, ki trpijo za nespečnostjo, je bilo ocenjenih tudi s klinično depresijo. Več kot 50% depresivnih ljudi pa pravi, da ponoči slabo spijo [21].

Med opcijami za razvoj naše aplikacije je odprta tudi možnost uporabe s svetili LED, ki omogočajo nadzor uporabe določene svetlobe. Z uporabo teh svetil in pomočjo urnika, oziroma senzorja zunaj, bi lahko ugotovili katera svetloba je trenutno najbolj primerna. Tako bi med dnevom uporabljali modro svetlobo, ki bi naše počutje izboljšala. Ponoči bi to modro svetlobo ugasnili oziroma zamenjali z rdečo, ki nam je manj škodljiva [13].

Težava pri primernem uravnavanju svetlobe za ljudi je ta, da zelo hitro zaznamo spremembe pri majhnih vrednosti, pri višjih vrednostih pa so nam skoraj neopazne. Področje, ki se ukvarja s tem se imenuje psihofizika. Natančneje v tem primeru govorimo o Weber–Fechnerjevem zakonu. Ta je v tem diplomskem delu pomemben iz preprostega razloga. Močnejša svetloba je ljudem bolj škodljiva, vendar Weber–Fechnerjev zakon pravi, da bomo pri manjših intenzitetah razlike bolje opazili [19]. To pomeni, da v temnejšem primeru lahko moč svetlobe le rahlo dvignemo, prostor pa se nam bo zdel občutno svetlejši. Hrati to žal pomeni, da razlike nad nekim nivojem ne bomo več zaznali, svetloba pa nam bo vseeno bolj škodovala. Kot vidimo na grafu (slika: 3.1) je funkcija zakona logaritmične vrste. Razberemo lahko, da pri relativnem povečanju svetlobe za faktor 20, ljudje zaznamo kot da je prostor trikrat svetlejši, pri faktorju 40 pa le štirikrat. Splača se razmisliti, če bi se splačalo omejiti moč svetil. Na ta način bi zagotovili že precej dobro svetlobo, omejili vpliv negativnih učinkov umetne svetlobe in povrh vsega še zmanjšali električne stroške.



Slika 3.1: Fechnerjev zakon.

Poglavje 4

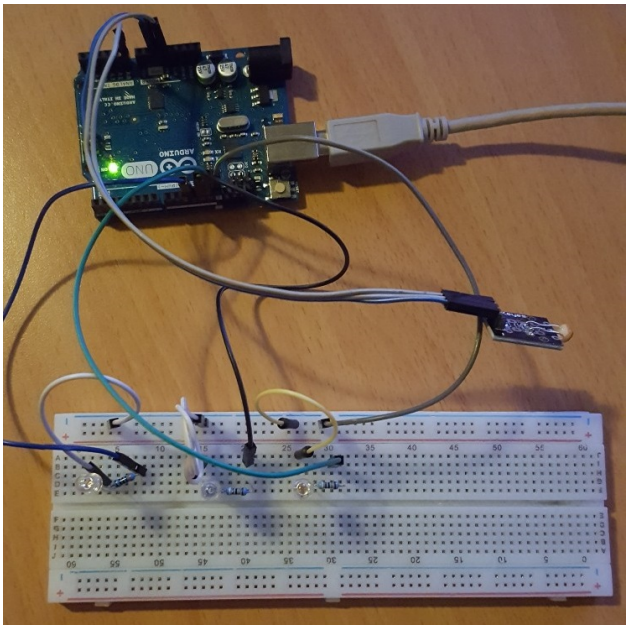
Implementacija

4.1 Vezje

Prototip implementacije sistema za nadzor svetlobe v prostoru smo začeli s pripravo vezja. Na naš Arduino UNO smo zvezali tri svetila LED in svetlobni senzor. Komponente smo na tej stopnji povezali na mikrokrmilnik preko testne plošče (ang. breadboard) zaradi večje preglednosti in lažje odpravljanje možnih napak. Svetlobni senzor je v resnici fotoupor z že zvezanim dodatnim uporom, tako da ima tri priključke namesto samo dveh. Eno izmed njih smo zvezali na napajanje, drugo ozemljili, zadnja pa je namenjena neposredni povezavi na analogni priključek mikrokrmilnika. Pri svetilih LED smo ploščate stranice zvezali na ozemljitev, napajanje zanje pa poteka preko 180 Ω upora in vsak na svoj analogni izhod. Vrednost potrebnega upora smo dobili po formuli [9]:

$$\frac{U_{\mathrm{S}} - U_{\mathrm{LED}}}{I},$$

kjer je $\mathrm{U_S}$ enak napetosti napajanja, $\mathrm{U_{led}}$ napetosti na svetilih LED in I tok v miliamperih. V našem primeru so bile te vrednosti $\mathrm{U_S} = 5$ V, $\mathrm{U_{LED}} = 1{,}7$ V in I = 20 mA. Tako pridemo do vrednosti 165 Ω. Upornik, ki ima upornost najbližjo tej vrednosti, pa je 180 Ω upornik.

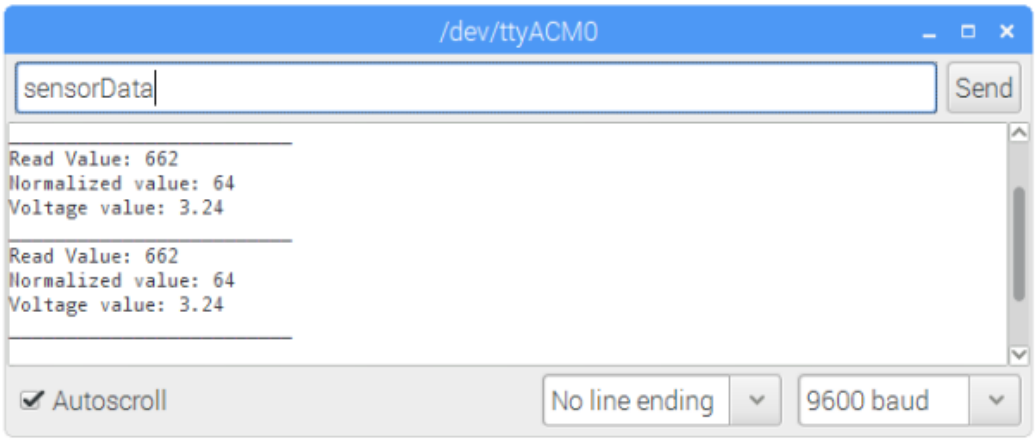


Slika 4.1: Arduino in testna plošča.

4.2 Arduino

Po opravljeni vezavi smo morali pripraviti kodo, ki bo to vezje nadzorovala. Ker smo se odločili za uporabo Raspberry Pi naprave za strežnik, smo preko te opravljali tudi pisanje kode na Arduino. Z ukazom „sudo apt-get install arduino“ smo na napravo naložili vse potrebno za programiranje mikrokrmilnika, vključno z Arduino IDE. Po pripravah Raspberry Pi smo se lotili programiranja. Za osnovno delovanje sistema so poleg osnovnih gradnikov „setup“ in „loop“ potrebne še štiri funkcije. Setup je potreben za deklaracijo in inicializacjo spremenljivk ter nastavitev vhodnih in izhodnih enot. Loop je glavni del kode, ki se izvaja ves čas. V njem preverjamo, če je mikrokrmilnik dobil kakšen ukaz. V primeru, da smo dobili ukaz, tega preberemo in posredujemo funkciji „parseString“, v kateri obdelamo ukaz in pokličemo nasledjo funkcijo z argumenti, pridobljenimi iz ukaza. Poklicana funkcija je lahko dveh tipov. En izmed njiju je branje vrednosti s senzorja „readValue“, drug je pisanje vrednosti na izhod, na katerem je priklopljeno svetilo LED

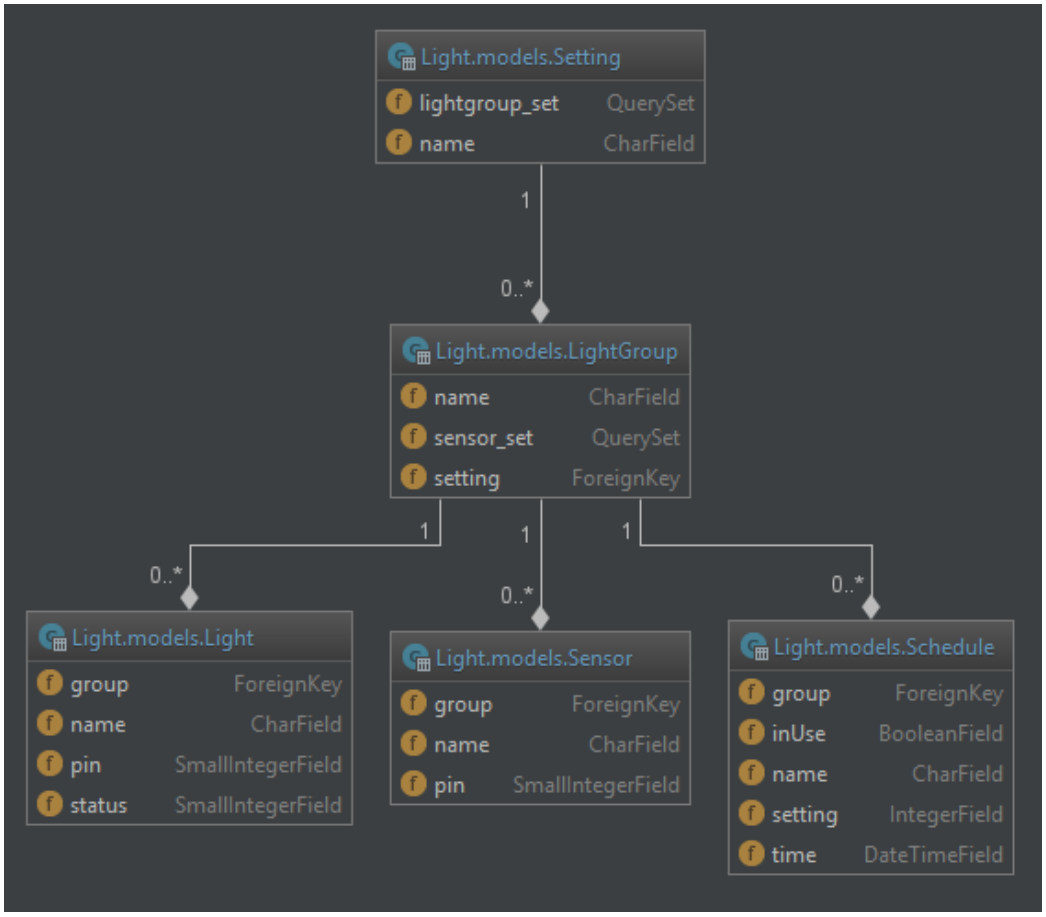
imenovana „writeValue“. V vsakem primeru po koncu sledi še preobložena funkcija imenovana „returnString“. Ta funkcija je namenjena pošiljanju povratne informacije. V primeru, da je prišla zahteva po vrednosti senzorja, jo normalizira v vrednost od 0 do 100 in pošlje, če je šlo za nadzor svetilke LED pa vrne samo „'True'“ v primeru uspešno izvedenega ukaza ali „'False'“ za neuspešno izvedeno funkcijo.Ta funkcija je tudi zadnja preden se ponovno začne izvajati „loop“ in poskrbi, da Raspberry Pi ve, kaj se je zgodilo po njegovem ukazu. Kot dodatno funkcionalnost smo dodali opcijo, da v primeru pomanjkanja analognih izhodov svetilke nadzorujemo preko digitalnih izhodov s pomočjo PWM tehnike. Ta deluje tako, da s spreminjanjem frekvence spreminjamo kako močno svetilo LED sveti [10]. Vse funkcije smo pretestirali z ročnim vnosom podatkov preko Serial Monitorja (slika 4.2), ki je vključen v Arduino IDE.



Slika 4.2: Testiranje v serijskem monitorju.

4.3 Podatkovna baza

Z zaključkom na Arduino UNO je sledilo programiranje API, ki bo z njim komuniciral. Začeli smo s postavitvijo podatkovne baze. Osnovno je sestavljena iz tabel *Light*, *Sensor* in *Group*. Kasneje smo dodali še tabeli *Setting* in *Schedule* (slika 4.3). Tabela za shranjevanje podatkov o uporabnikih že obstaja znotraj ogrodja Django. *Light* vsebuje podatke o svojem imenu za lažje prepoznavanje, številko pina na katerega je priključen, trenutni svetlosti in kateri skupini pripada. Vsebuje tudi metode, s katerimi lahko te podatke dodajamo, odstranimo ali samo spremenimo. Tabela *Sensor* je sestavljena podobno, razlika je odsotnost trenutne svetlosti saj tega podatka ne bomo shranjevali. Če bomo te informacije potrebovali, jih bomo pridobili preko mikrokontrolerja. Zadnja tabela je namenjena grupiranju senzorjev in luči za samostoječi nadzor. Med implementiranjem sistema smo dodali še tabelo *Settings*. V njej hranimo podatke o tem kakšno nastavitev skupina trenutno uporablja. Privzete nastavitve so *0* za izključena svetila, *1* za ročni nadzor, *2* za samodejni nadzor, *3* za kombinacijo ročnega in samodejnega nadzora. Tabela *Schedule*, ki je bila dodana zadnja, hrani podatke o skupini in katere nastavitve naj se uporabijo ob določeni uri. Obstajala je opcija, da bi nastavitve tabele Schedule prepisale nastavitev v skupini, zato smo dodali polje *inUse*, ki aplikaciji pove, če urnik trenutno uporabljamo. API dnevno preveri tabelo *Schedule* in shrani podatke, ki jih mora izvesti. Prav tako te podatke posodobi vsakič, ko zazna spremembo v dotični tabeli. Četrti način deluje tako, da je samodejen nadzor vključen čez dan, po sončnem zahodu pa se priklopi na točni nadzor.



Slika 4.3: ER diagram

Ker smo se odločili za uporabo ogrodja Django, smo seveda tudi bazo postavili z njegovo pomočjo. Django uporablja arhitekturo MVT, kjer M stoji za model. Tabele se v tem ogrodju gradijo iz modelov, ki smo jih napisali in se shranijo v jeziku „SQL Data Engine“ naše izbire. S prostorom in močjo naprave poskušamo varčevati, zato smo izbrali SQLite3. V primeru „class Light“ lahko vidimo kako je sestavljen model za tabelo „Light.models.Light“ na sliki 4.3.

```
class Light(models.Model):
    # Name of our light
    name = models.CharField(max_length=32)

    # Power represents intensity of light
    status = models.SmallIntegerField(
        validators=[MinValueValidator(0), MaxValueValidator(100)]
    )
    pin = models.SmallIntegerField(
        validators=[MinValueValidator(0), MaxValueValidator(13)],
        unique=True
    )
    group = models.ForeignKey(
        LightGroup,
        models.SET_NULL,
        blank=True,
        null=True,
    )

    @classmethod
    def create(cls, name, pin):
        light = cls(name=name, pin=pin, status=0)
        return light

    def __str__(self):
        return self.name + ": " + str(self.status)
```

4.4 API in aplikacija

S postavitvijo baze smo bili pripravljeni na programiranje API dela sistema. Urediti smo morali komunikacijo z Arduino UNO, podatkovno bazo in seveda sprogramirati funkcje za komunikacijo z aplikacijami, ki bodo do našega sistema dostopale. Brez komunikacije z mikrokontrolerjem tudi ostalih funkcij ni mogoče testirati, zato smo se prvo lotili tega dela. Potrebni sta samo dve funkciji, in sicer ukaz za podatke senzorja in ukaz za zapis vrednosti na izhod.

Za nadzor več svetil sočasno smo uporabili v bazi definirane skupine. Pri ročnem vnosu vrednosti to pomeni, da določimo skupino in moč s katero naj svetijo članice skupine. API z dobljenimi podatki izvede poizvedbo nad podatkovno bazo, da najde vsa svetila v izbrani skupini. Svetilom nato spremeni vrednost moči v podatkovni bazi ter pošlje podatke o spremembi Arduino, tako da se vrednost spremeni tudi na svetilu samem. Ker nočemo, da bi lahko več skupin vplivalo na eno svetilo, smo to opcijo prepovedali. Tako je lahko v skupini več svetil, svetilo pa je lahko v le eni skupini.

Samostojen nadzor svetlobe nadzira svetila na podoben način kot ročen vnos. Razlika je v tem, da za spremembo ne potrebuje naše aktivnosti. Spreminja se namreč glede na vrednosti prebrane na svetlobnem senzorju. Vrednosti svetil se spreminjajo samo glede na senzorje, ki so v isti skupini kot so svetila. Samodejno spreminjaje svetlobe se prilagaja glede na najvišjo vrednost, ki jo senzorji preberejo. Vsem svetilom nato dvignejo ali znižajo moč tako, da je le nekaj odstotkov višja od senzorjeve prebrane vrednosti. Odločili smo se, da je za testno obdobje dovolj, če omogočimo, da je lahko vsako svetilo v samo eni skupini, tako se izognemo težavam prepisovanja stanja.

Naslednja naloga je bila priprava pogleda, ki uporabniku omogoča dodajanje, spreminjanje in brisanje svetil ter senzorjev. Metode za spreminjanje podatkov v podatkovni bazi smo že pripravili. Čakalo nas je le še povezovanje podatkovne baze preko APIja s pogledom. To storimo tako, da naredimo predlogo vnosnega obrazca (ang. form).

```
<div class="container-fluid">
    <form action="light_add.html" method="post">
        {% csrf_token %}
        {{ form }}
        <input type="submit" value="submit">
    </form>
</div>
```

Zgornji obrazec lahko kasneje uporabimo tudi v drugih delih kode, če bo to potrebno. To je možno zaradi visoke modularnosti ogrodja Django, ki nam omogoča, da kodo razbijemo na veliko neodvisnih delov. Za obrazcem moramo napisati kodo, ki bo poskrbela, da bo predloga vedela katera polja mora vsebovati. V ta namen v datoteko „forms.py“ napišemo meta razred prikazan spodaj.

```
class LightAddForm(ModelForm):
    class Meta:
        model = Light
        fields = ['name', 'pin']
```

Vsi ti kosi programa se nato povežejo v „view.py“. V naslednjem primeru vidimo, da v primeru, ko še nimamo podatkov, sestavi stran iz vseh do sedaj omenjenih kosov. Če smo podatke že vpisali, pa jih shrani preko funkcije, ki uporabi metodo v prej omenjenih „models.py“.

```
@login_required
def light_add(request):
    form = LightAddForm()

    if request.method == 'POST':
        name = request.POST['name']
        pin = int(request.POST['pin'])
        light_add_to_db(name, pin)
        return HttpResponse("Light Saved")

    return render(request, 'light_add.html', {'form': form})
```

Z zgornjim delom kode smo zaključili delo za dodajanje novega svetila v sistem. Postopek za urejanje in brisanje je precej podoben. Enako velja za funckije in metode potrebne za delo z senzorji in skupinami. Na spodnjem primeru je primer sporočila JSON, ki ga dobimo, kadar potrebujemo podatke o svetilu.

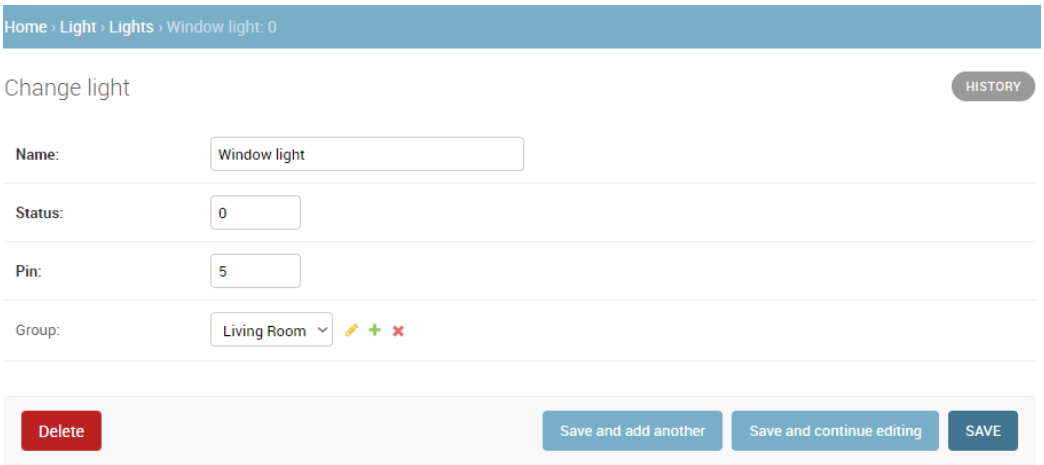
```
[
    {
        "model": "Light.light",
        "pk": 1,
        "fields": {
            "name": "Window light",
            "status": 0,
            "pin": 5,
            "group": 2
        }
    }
]
```

Django nam omogoča tudi uporabo strani Django Admin. Na tej strani lahko z uporabniškim imenom in geslom dostopamo neposredno do podat-

kov v podatkovni bazi. Preden se podatki pokažejo, jih moramo dodati v „admin.py“, kot je prikazano spodaj. Podatke lahko nato spreminjamo neposredno, brez uporabe naše aplikacije. Ta pristop smo uporabili že pri testiranju APIja, preden smo dokončali našo aplikacijo.

```
from django.contrib import admin
from .models import *
# Register your models here.
admin.site.register(Light)
admin.site.register(Sensor)
admin.site.register(Schedule)
admin.site.register(LightGroup)
admin.site.register(Setting)
```

Na sliki 4.4 vidimo kako Djangova administrativna stran prikaže podatke, ki smo jih predstavili v zgornjem JSON primeru.

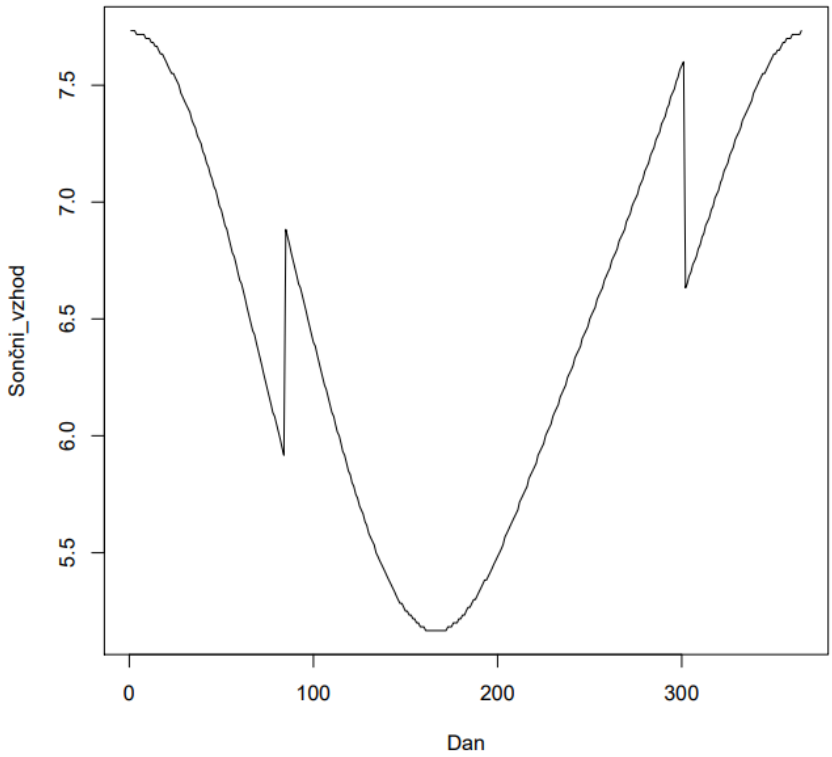


Slika 4.4: Podatki JSON uporabljeni v administraciji Django.

4.4.1 Samodejno uravnavanje

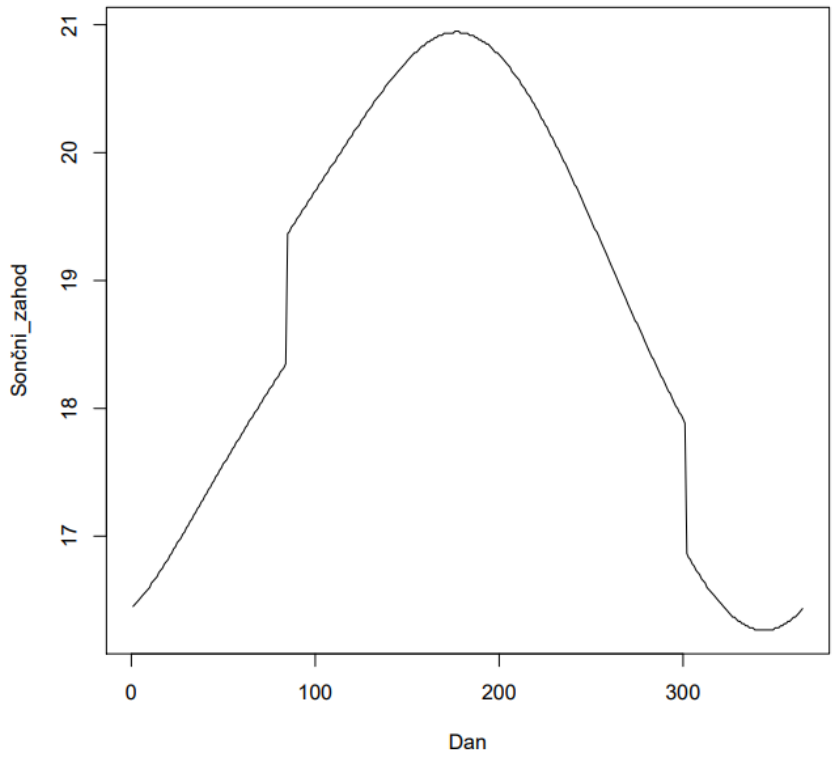
Med glavnimi cilji tega diplomskega dela je bil sistem, ki se lahko uravnava sam s pomočjo senzorjev. Naš cilj je bil sistem, ki lahko sam preračuna primerno svetlobo, glede na zunanje vire. Jesen je v povprečju temnejša kot poletje, zato je tudi v notranjosti potrebne manj svetobe, da prostor zaznamo kot svetlega. V implementaciji smo to upoštevali tako, da smo za računanje primerne svetlosti vzeli svetlost zunaj objekta in notranjost osvetlili le za nekaj odstotkov. Na ta način očem ne škodujemo s prevelikimi spremembami v intenziteti svetlobe. Tako tudi poskrbimo da imajo oči minimalno dela s prilagajanjem. V sistem smo želeli implemetirati metode za nadzor modre svetlobe skozi dan. V tretjem poglavju smo omenili rezultate nekaj raziskav opravljenih v preteklem desetletju. Med njimi so bile pogoste ugotovitve, da modra svetloba na ljudi pozitivno vpliva čez dan in negativno ponoči. Problema smo se nameravali lotiti tako, da bi uporabljali svetila LED, pri katerih je možnost nadziranja barve svetlobe. Na žalost nam implementacija ni uspela, saj smo ugotovili, da bi morali spremeniti večji delež že postavljenega sistema. Razlog za to, je napaka pri analizi zahtev.

Čeprav nam ni uspelo implementirati nadzora modre svetlobe, smo z implementacijo dosegli nekaj drugega. Skozi leto se čas sončnega vzhoda in zahoda spreminjata, prav tako se dvakrat na leto premakne ura. Če bi bil naš sistem sestavljen le na ročni vpis ali vezan na uro, bi imeli velike težave s prilagajanjem na sonce skozi leto. Naša implementacija pa se je težave uravnavanja svetlobe v prostoru lotila s senzorji, kar pomeni da nismo vezani na vreme, letni čas ali karkoli drugega. Na sliki 4.5 imamo graf, ki prikazije nihanje sončnega vzhoda skozi leto.



Slika 4.5: Sončni vzhod skozi leto.

Podatki grafov za sončni vzhod (slika: 4.5) in zahod (slika: 4.6) so na voljo na spletu [14]. Grafa pa smo ustvarili v programskem jeziku R [7]. Po pregledu teh grafov smo se odločili, da v aplikacijo dopišemo funckijo, ki pri preverjanju senzorjev dodatno preveri, če so vrednosti pod določenim pragom. V primeru, da se je to zgodilo, se sistem preklopi na ročni način. Ko se zgodi obratno, torej vrednosti na senzorju pridejo nad minimalen prag, se sistem preklopi nazaj na samodejno vodenje. Na ta način zagotovimo, da je naš sistem res neodvisen od ure in dolžine dneva.



Slika 4.6: Sončni zahod skozi leto.

4.5 Varnost

Za zaključek moramo poskrbeti še za varnost pri dostopu do naše aplikacije. To dosežemo tako, da povsod kjer dostopamo do branja in pisanja podatkov, uporabimo žeton CSRF. Ogrodje Django vsebuje značko {% csrf_token %}, ki prepreči, da bi nekdo uporabil overilnice, ki so jih ukradli med obiskom okužene strani [23]. Prav tako smo dodali na vse strani značko @login_required, s katero preprečujemo, da bi neprijavljen uporabnik dostopal do katerekoli strani.

Poglavje 5

Možne nadgradnje

Ker je sistem narejen preko APIja zelo modularen, so odprte tudi mnoge možnosti razširitve.

5.1 Senzor IR

S senzorjem IR lahko nadgradimo naš sistem za nadzor svetlobe. To storimo tako, da sproti preverjamo še, če je v prostoru kakšna oseba. To nam omogoča, da se v sistemu ugasnejo svetila, če ne zaznavajo gibanja. V primeru, da je soba nastavljena na ročno prižiganje in ugašanje namesto samodejno, lahko s pomočjo senzorja IR od doma izvemo, če se trenutno kdo giba po prostoru. Tako lahko spremenimo stanje svetlobe v prostoru, ko nas ni doma, ne da bi nas skrbelo, da smo komu ugasnili luči, medtem ko je v prostoru. Naslednja opcija, ki bi jo lahko dogradili je, da vgradimo v vsak prostor več senzorjev, tako da lahko spremljamo pozicijo oseb. S tem lahko prižigamo samo svetila katerim smo blizu. Svetila, ki so oddaljena, pa lahko medtem zatemnimo. Poleg nadgradnje svetlobnega dela sistema, bi lahko te senzorje uporabili tudi za varovanje prostorov. Sensor IR lahko deluje kot alarm pred neželjenimi gosti, saj nam lahko sistem sporoči, da se je zgodilo nepričakovano gibanje v prostoru.

5.2 Toplotni senzor

Arduino prav tako podpira temperaturne senzorje. Z uporabo temperaturnega senzorja tipa TMP36 [15], lahko preprosto merimo temperaturo prostora. Če priključimo senzor na 3 V, lahko podatke preračunamo v °C po sledeči formuli $°\mathrm{C} = (U - 0.5\mathrm{V}) * 100.0\frac{°\mathrm{C}}{\mathrm{V}}$ Čeprav ni natančen na decimalko, lahko to ignoriramo zaradi nizke cene. Če želimo bolj natančne meritve, lahko uporabimo dražji čip in temu primerno prilagodimo formulo. Podatke lahko beremo z istimi funkcijami, ki smo jih uporabljali za branje svetlobnih senzorjev, paziti moramo le na to, da v bazi pravilno povemo katerega tipa je senzor, tako da lahko prebrane podatke pravilno obdelamo. Z dodatno nadgradnjo sistema bi lahko nadzirali tudi gretje in hlajenje prostora. Žal je to že preširoka tema, da bi jo lahko obdelali v tem diplomskem delu.

5.3 Senzor vlage

DHT11 je Arduinov senzor, ki podpira branje temperature in vlage v prostoru [13]. Ker je branje relativne vlage v prostoru odvisno tudi od temperature, moramo tu uporabiti bolj napreden senzor, ki je sposoben obojega. Uporaba bolj naprednega senzorja nam tudi olajša branje podatkov, saj so podatki že takoj v človeku razumljivi obliki. Enako kot pri senzorju TMP36 pa velja, da ga lahko v sistemu uporabimo brez nadgradenj sistema, le v tabelo moramo dodati nov tip senzorja. V primeru, da bi nadgradili podatkovno bazo in API, bi lahko z vrednostmi DHT11 naredili veliko zanimivih funkcij. Poleg tega, da v prostoru ohranjamo optimalno vlago s pomočjo razvlaževalnikov, je mogoče nadzorovati tudi vlago v zemlji, kar nam lahko pomaga pri oskrbi raznih rastlin. Če želimo, si lahko omislimo celo savno, ki bi jo nadzorovali s tem sistemom.

5.4 Senzor za dim

Senzor za dim je izjemno lahko vključiti v naš sistem. Njegova edina funkcija je, da preverja za količino dimnih delcev. Naš sistem te podatke bere brez težav, saj imamo že vse potrebne funkcije. Obdelava podatkov pa je še bolj preprosta kot pri ostalih senzorjih, saj moramo samo preverjati, če so senzorji v mejah normale. V nasprotnem primeru sporočimo uporabniku, oziroma določeni napravi, da gori.

5.5 Primeri uporabe z nadgradnjami

V primeru da bi sistem nadgradili z vsemi temi senzorji, se možnosti uporabe hitro dvignejo. Prvi način je bil že omenjen pri zalivanju rož. S pravilnim pristopom lahko naš sistem brez težav uporabimo za upravljanje tople grede ali drugih načinov vzgajanja raznih rastlin. Ker lahko prosto nadziramo svetlobo, vlago in temperaturo, bi lahko vzgajali praktično katerokoli vrsto rastline od puščavskih do tropskih vrst. Poleg vzgajanja rastlin, bi lahko vzrejali tudi živali. Čeprav bi lahko nadzirali velike objekte kot je hlev, se bomo tokrat spustili na manjši nivo. Terarij za male živali. Mali tropski ljubljenčki so vedno bolj popularni, vendar ljudje preprosto nimajo znanja ali časa, da bi zanje pravilno skrbeli. S pomočjo sistema bi samo vpisali zahtevane vrednosti in senzorji bi sami poskrbeli, da bi terarij ponujal podobno življensko okolje, kot ga ima žival v naravnem življenskem okolju. Za sistem imamo še mnogo možnosti kako bi ga lahko dodatno razvili, vendar se bomo za zdaj ustavili tu.

Poglavje 6

Sklep

Diplomskega dela smo se lotili z namenom izdelave prototipa sistema za uravnavanje svetlobe v prostoru. Na trgu je trenutno veliko ponudnikov, ki ponujajo storitve pametne razvsvetljave. Težava pri teh je visoka cena, saj se ti ponudniki zanašajo na uporabo pametnih žarnic, ki dobijo vsaka svoj IP naslov, preko katerega jih nadzirajo. Naša rešitev se je težave lotila z mikrokontrolerjem, na katerega povežemo svetila, ki jih želimo nadzirati. Prototip sistema deluje zadovoljivo, vendar smo tekom dela ugotovili, da bi se pri dejanski implementaciji v stanovanju zapletlo. Stroški so v osnovi zelo majhni, vendar bi za dejansko implementacijo potrebovali zatemnilnike, ki bi nadzorovali moč svetil. Cena tako preraste stroške, ki bi si jih ustvarili z uporabo žarnic s svojim spletnim naslovom. Preostanek sistema pa je medtem dosegel naša pričakovanja. Tudi če se odločimo spremeniti delovanje, tako da bi uporabili žarnice s svojim naslovom, lahko to storimo z minimalnim trudom.

Prednosti naše aplikacije:

- visoka modularnost,
- prilagodljivost,
- varnost,
- preprosta nadgradljivost.

Slabosti:

- visoka cena implementacije,
- ni samodejnega zaznavanja novih senzorjev in svetil.

Čeprav prototip morda ni najbolj primeren za predelavo bivalnega prostora, bi bil odličen za izdelavo manjših modulov. Z nekaj nadgradnjami, ki smo jih opisali v prejšnjih poglavjih, ima zelo visok potencial za nadzor pametnih terarijev, toplih gred in podobnih biomskih aplikacij.

Menimo, da je diplomsko delo kvalitetno opravljeno. Čeprav s cenovnega pogleda ni najbolj optimalen, ima veliko večji potencial za nadaljni razvoj.

Literatura

[1] Arduino IDE. Dosegljivo: https://www.arduino.cc/en/Main/Software. [Dostopano: 29. 11. 2017].

[2] Arduino loop dokumentacija. Dosegljivo: https://www.arduino.cc/reference/en/language/structure/sketch/loop/. [Dostopano: 29. 11. 2017].

[3] Arduino Uno Rev3. Dosegljivo: https://store.arduino.cc/usa/arduino-uno-rev3. [Dostopano: 28. 11. 2017].

[4] Blue light has a dark side. Dosegljivo: https://www.health.harvard.edu/staying-healthy/blue-light-has-a-dark-side. [Dostopano: 3. 12. 2017].

[5] Bootstrap dokumentacija. Dosegljivo: https://getbootstrap.com/docs/4.0/getting-started/introduction/. [Dostopano: 29. 11. 2017].

[6] CSS reference. Dosegljivo: http://devdocs.io/css/. [Dostopano: 29. 11. 2017].

[7] Domača stran R. Dosegljivo: https://www.r-project.org. [Dostopano: 4. 12. 2017].

[8] JavaScript Object Notation. Dosegljivo: https://www.json.org. [Dostopano: 29. 11. 2017].

[9] Ohmov zakon. Dosegljivo: http://www.electronics-tutorials.ws/dccircuits/dcp_2.html. [Dostopano: 29. 11. 2017].

[10] Pulse Width Modulation. Dosegljivo: https://www.arduino.cc/en/Tutorial/PWM. [Dostopano: 29. 11. 2017].

[11] PyCharm IDE. Dosegljivo: https://www.jetbrains.com/pycharm/features/. [Dostopano: 29. 11. 2017].

[12] RASPBERRY PI 3 MODEL B. Dosegljivo: https://www.raspberrypi.org/products/raspberry-pi-3-model-b/. [Dostopano: 29. 11. 2017].

[13] Senzor DHT11. Dosegljivo: https://akizukidenshi.com/download/ds/aosong/DHT11.pdf. [Dostopano: 29. 11. 2017].

[14] Spletna stran timeanddate. Dosegljivo: https://www.timeanddate.com. [Dostopano: 4. 12. 2017].

[15] Toplotni senzor TMP36. Dosegljivo: http://www.analog.com/media/en/technical-documentation/data-sheets/TMP35_36_37.pdf. [Dostopano: 29. 11. 2017].

[16] A. Bahga and V. Madisetti. *Internet of Things: A Hands-On Approach:* . Arsheep Bahga & Vijay Madisetti, 2014.

[17] Steve Bush. Dongle computer lets kids discover programming on a TV. Dosegljivo: https://www.electronicsweekly.com/market-sectors/embedded-systems/dongle-computer-lets-kids-discover-programming-on-a-2011-05/, 2011. [Dostopano: 28. 11. 2017].

[18] Rory Cellan-Jones. A 15 pound computer to inspire young programmers. Dosegljivo: http://www.bbc.co.uk/blogs/thereporters/rorycellanjones/2011/05/a_15_computer_to_inspire_young.html, 2011. [Dostopano: 28. 11. 2017].

[19] Gustav Fechner. Elements of psychophysics. Vol. I. 1966.

[20] Roy T Fielding and Richard N Taylor. *Architectural styles and the design of network-based software architectures*. University of California, Irvine Doctoral dissertation, 2000.

[21] Megumi Hatori, Claude Gronfier, Russell N Van Gelder, Paul S Bernstein, Josep Carreras, Satchidananda Panda, Frederick Marks, David Sliney, Charles E Hunt, Tsuyoshi Hirota, et al. Global rise of potential health hazards caused by blue light-induced circadian disruption in modern aging societies. *npj Aging and Mechanisms of Disease*, 3(1):9, 2017.

[22] Amy Nordrum. Light Fountain—an interactive art installation. Dosegljivo: `https://spectrum.ieee.org/tech-talk/telecom/internet/popular-internet-of-things-forecast-of-50-billion-devices-by-2020-is-outdated`, 2016. [Dostopano: 25. 11. 2017].

[23] OWASP. CSRF. Dosegljivo: `https://www.owasp.org/index.php/Cross-Site_Request_Forgery_(CSRF)`. [Dostopano: 29. 11. 2017].

[24] Peter Price. Can a £15 computer solve the programming gap? Dosegljivo: `http://news.bbc.co.uk/2/hi/programmes/click_online/9504208.stm`, 2011. [Dostopano: 28. 11. 2017].

[25] W3C. HTML 5.1 2nd Edition. Dosegljivo: `https://www.w3.org/TR/html51/index.html#contents`. [Dostopano: 29. 11. 2017].